Relazione per “Goose Game”

Leonardo Carboni - Andrea Ongaro
Davide Denicolò - Endri Domi

Anno Accademico 2020/2021

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il software, realizzato come elaborato di "Programmazione ad Oggetti" del corso di Ingegneria e Scienze Informatiche dell'Università di Bologna per l'anno accademico 2020-2021, ha come obiettivo la realizzazione di una variante del gioco dell'Oca.
Il gioco presenta, rispetto a quello originale, un maggior coinvolgimento dell'utente grazie alla presenza di minigiochi che permettono di avanzare o di retrocedere a seconda dell'esito del minigioco.
L'obiettivo del gioco è quello di arrivare alla casella finale, ma ci sarà una difficoltà aggiuntiva ovvero quella di dover arrivare esattamente sull'ultima casella e, nel caso in cui la si superi, si tornerà indietro del numero di caselle superate.

Requisiti Funzionali

- Potranno essere effettuate partite tra due, tre o quattro giocatori in locale.
- Ogni giocatore potrà creare un proprio nome o decidere di scegliere il nome dalla lista di nomi già utilizzati in precedenza e il proprio colore tra quelli disponibili.
- Lancio di mini giochi a seconda della posizione del giocatore.
- Salvataggio su file della classifica dei giocatori al termine della partita.

Requisiti non funzionali

- Realizzazione di una grafica minimale e semplice per poter essere usufruita facilmente da persone di tutte le età.
- Fluidità mini giochi.

1.2 Analisi e modello del dominio

Il giocatore potrà lanciare un dado per potersi muovere all'interno del campo di gioco, il quale sarà costituito da 42 caselle (comprese quelle di inizio e fine). Ci saranno due tipi diversi di caselle, le prime dove non capiterà niente al giocatore mentre le seconde apriranno un mini gioco. I mini giochi saranno otto e ognuno avrà una logica diversa. Verrà inoltre visualizzata una classifica in base alla posizione dei giocatori. Dunque, le entità rilevate sono le seguenti:

- Il giocatore che avrà un nome e un colore identificativo. Il nome dovrà essere univoco.
- Un dado tradizionale che, quando lanciato, restituirà un valore compreso tra 1 e 6
- Una casella che potrà essere di due tipi diversi
- Il campo di gioco che conterrà l'insieme delle caselle.
- La classifica dei giocatori

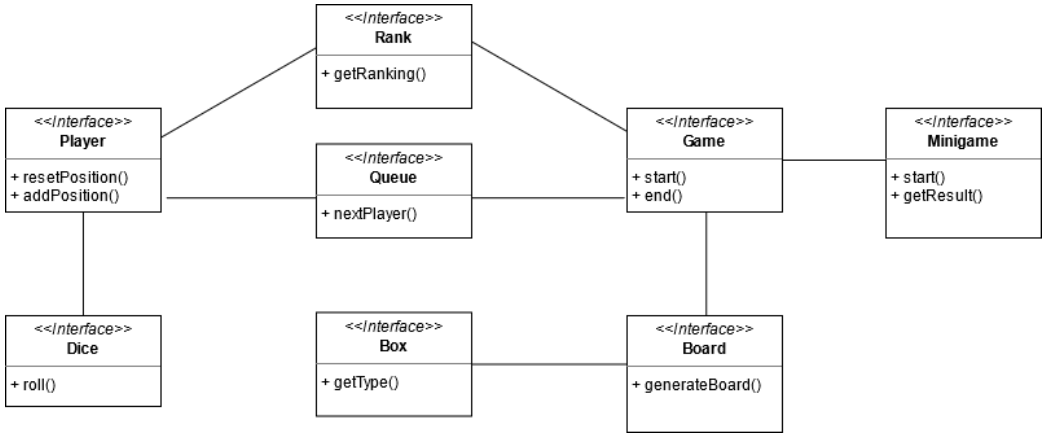


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

La realizzazione dell'applicativo è stata completata utilizzando il pattern architetturale MVC. Questo pattern è composto da tre componenti:

- Model: gestisce i dati, la logica e le regole dell'applicazione
- View: presenta i dati all'utente
- Controller: riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

Sfruttando appieno questo pattern si riesce a favorire un disaccoppiamento tra la logica applicativa e l'interfaccia utente favorendo una più facile implementazione di feature future in caso di cambiamento della grafica utente.
Il nostro software sfrutta il modello racchiudendo tutte le entità rilevate in fase di analisi e le logiche dei minigiochi nel model. La sezione della view contiene tutte le GUI dei vari menu e delle interfacce realizzate per i minigiochi. Infine il controller comprenderà tutte le classi che si occupano di gestire le interazioni tra le viste il model. Di conseguenza verrà implementata una classe controller per ogni vista utente.
Il controller si occupa di creare la view e dopo il setup iniziale, mostrandola tramite il metodo showAndWait() della classe Stage. A quel punto la classe chiamante viene bloccata fino al termine del processo della classe figlia. Per evitare problemi di consistenza e presenza di thread orfani si è deciso di implementare un bottone in ogni minigioco che viene abilitato solo a gioco finito. Cliccato il bottone viene ripassato il controllo alla classe chiamante (mainGame) che si occupa di ottenere il risultato. Si è scelto di procedere in

questo modo poiché gli Observer sono stati deprecati da Java 9 e i bus sarebbero stati troppo elaborati per una comunicazioni unidirezionale sincrona.
L'indipendenza tra il controller e la view non è completamente totale in quanto se venisse utilizzata con altre librerie grafiche potrebbe comportare la modifica dei listener.

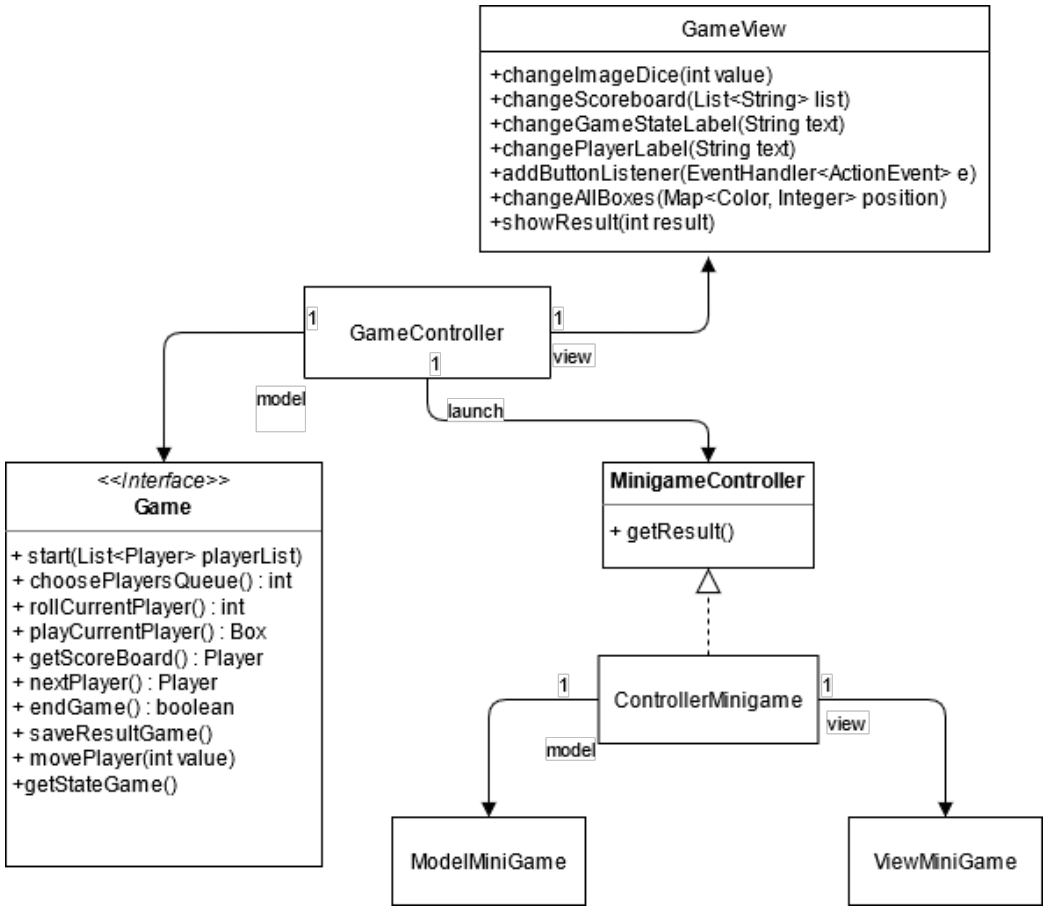


Figura 2.1:

2.2 Design dettagliato

Grazie alla divisione dei compiti ogni studente ha avuto modo di utilizzare in modo completo il pattern architetturale principale (MVC).

Leonardo Carboni

Menu

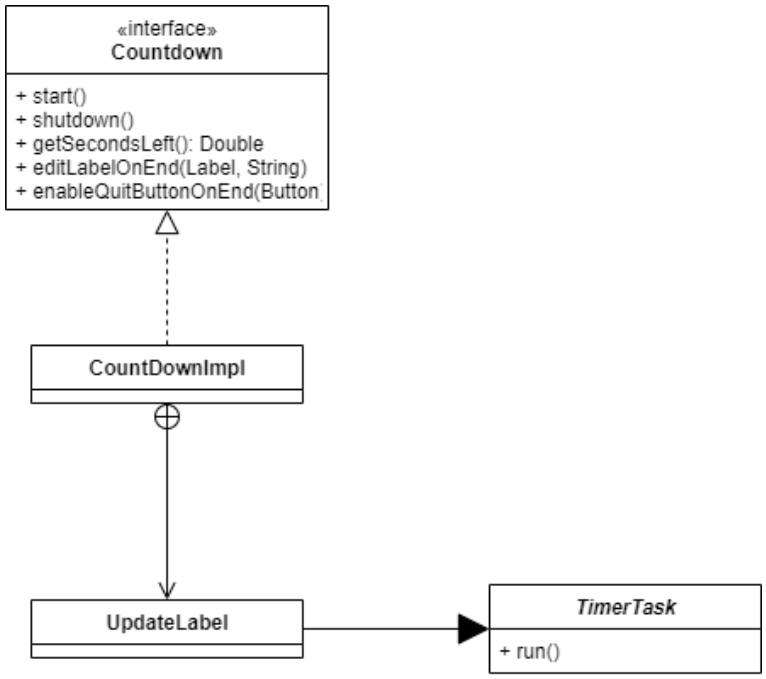
Il menu principale è composto da tre semplici bottoni, uno per avviare il gioco, uno che mostra le regole del gioco e l'altro per uscire.

PlayerChooser

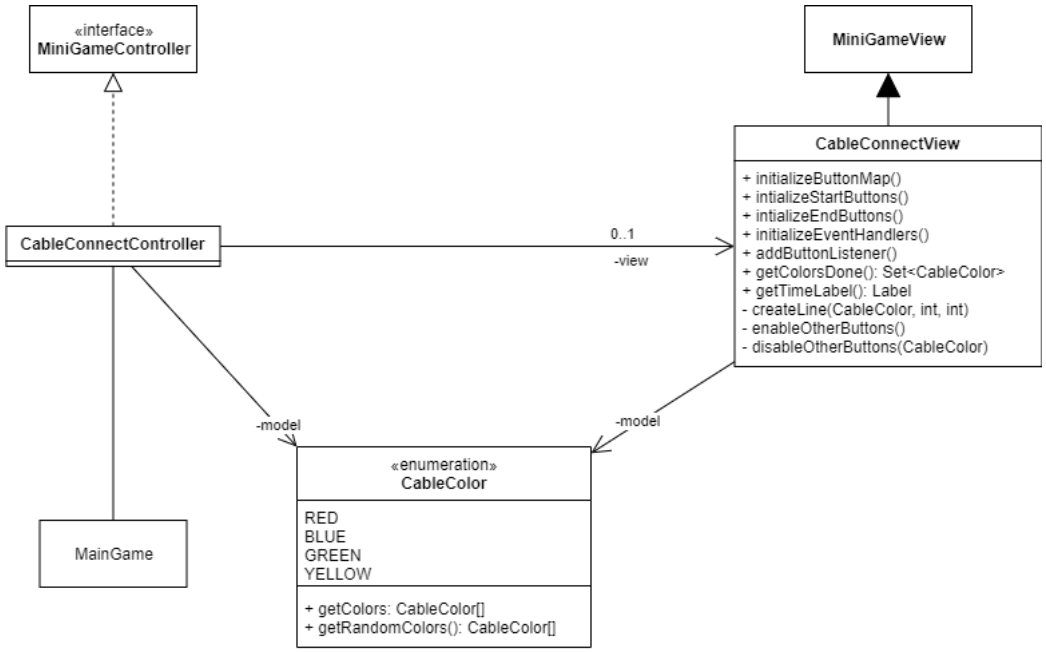
Interfaccia per la scelta dei personaggi, con le quattro pedine colorate sempre visibili. Vengono selezionate solo quelle con un nome e viene verificato che i nomi siano univoci. Andrea Ongaro si è occupato di salvare sul file players.json i nomi di tutti i giocatori in modo da renderli disponibili sempre tramite menu a tendina (ComboBox).

CountDown

Classe di utility che istanzia un timer al quale viene passata una inner class UpdateLabel che implementa la funzione run() della classe astratta TimerTask, aggiornando la label passata al costruttore di countdown ogni decimo di secondo. è presente un metodo editLabelOnEnd(Lable, String) che al termine del countdown modifica una label, utile per il minigioco Memory e un metodo enableQuitButtonOnEnd(Button) che permette di attivare il bottone quitButton all termine del countdown nei minigiochi cableconnect e phrasecatch.

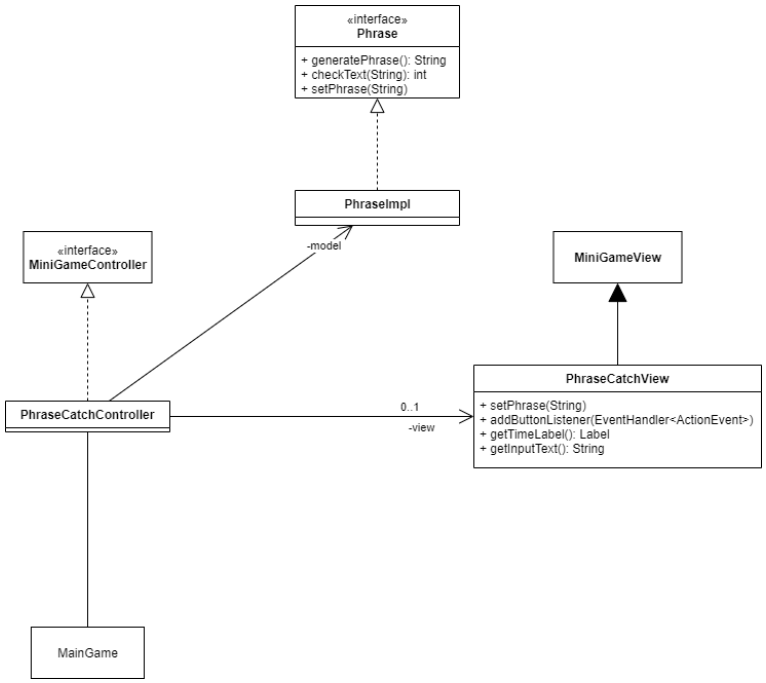


CableConnect
Minigioco nel quale bisogna collegare i "cavi" tramite pressione di bottoni. Risulta abbastanza semplice ma è stato scelto per ampliare le conoscenze di grafica dinamica, Il file FXML è composto da una label per il tempo restante e otto bottoni colorati in ordine casuale (quattro per lato). Alla pressione di un bottone colorato sul lato sinistro (lato di partenza) viene creato il punto di inizio di una linea tramite l'utilizzo di javafx.scene.shape.Line tramite coordinate x e y. Al movimento del mouse vengono modificate invece le coordinate di fine. L'aggiornamento di queste ultime viene terminato quando viene cliccato il bottone dello stesso colore nel lato destro. Durante il tracciamento viene abilitato solo il bottone dello stesso colore sul lato destro, mentre gli altri vengono disabilitati e settati a opacità 1, in modo che sembrino abilitati ma rimangano impossibili da cliccare. Al termine dell'ultimo collegamento il countdown viene fermato e viene abilitato il bottone quit, che permetterà di restituire il risultato, ovvero il numero di secondi interi rimanenti, al main game.



PhraseCatch:
Lo scopo del minigioco è scrivere la frase mostrata il prima possibile, il risultato viene calcolato in base agli errori fatti (malus) e ai secondi rimanenti (bonus). La frase viene scelta casualmente da un file sentences.txt, trovato

su internet e ridotto a frasi di lunghezza appropriata. Per problemi di creazione del file jar è stato usato un InputStream che legge il file tramite "Thread.currentThread().getContextClassLoader().getResourceAsStream()", lo porta in un InputStreamReader che viene usato in un BufferedReader, il quale buffer viene trasformato in una lista di stringhe dalla quale ne viene selezionata una casualmente. La classe phrase (model), dopo aver scelto una frase, tramite il metodo checkPhrase(String inputPhrase) verifica la corrispondenza di ogni carattere in base alla posizione. Alla pressione del tasto submit viene calcolato il risultato e attivato il bottone quit.



Duration
Semplice classe che trasforma la durata in millisecondi in una stringa di formato "HH:MM:SS".

WinScreen
Schermata semi statica che mostra la lista dei giocatori ordinati in base al punteggio e la durata della partita. Viene usata un metodo ricorsivo che mostra i nomi tramite un effetto di fade-in utilizzando la classe javafx.animation.FadeTrasnition.

Andrea Ongaro

Sequenza di Giocatori

La gestione della sequenza di giocatori è stata realizzata tramite la classe Queue. Essa implementa all'interno di sè un iteratore di player in modo tale che ci sia sempre un giocatore successivo. Vi è inoltre un metodo fondamentale per classe che si occupa di effettuare una copia profonda di una lista (una copia reale degli oggetti) evitando così problemi quando la lista di giocatori passati alla classe cambi.

Creazione Campo di Gioco

La creazione del campo di gioco pone le basi sulla gestione dei tipi di caselle che è stata realizzata tramite due tipi di enumerazioni. La prima si occupa di gestire i tipi generali di caselle che, in questo caso, sono due, ma non esclude la possibilità di aggiungerne altre in future versioni. Un tipo viene associato a caselle che non comportano azioni mentre il secondo presenta La seconda, invece, raccoglie tutti i tipi effettivi di caselle, la casella di partenza, quella di fine e quelle di tutti i tipi di minigiochi. Ogni casella ha all'interno di se stessa il tipo generale a cui fa riferimento. Dopo aver gestito le caselle è stato affidata la creazione del campo di gioco ad una classe dedicata che gestisce il numero minimo di caselle e ogni quanto devono essere presenti ogni tipo di casella.

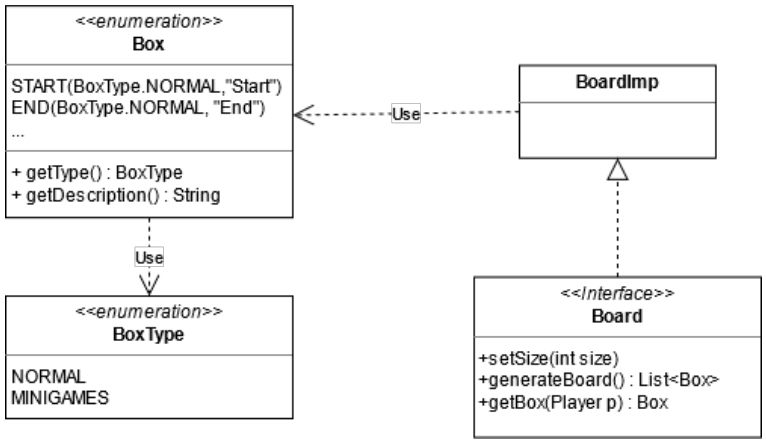
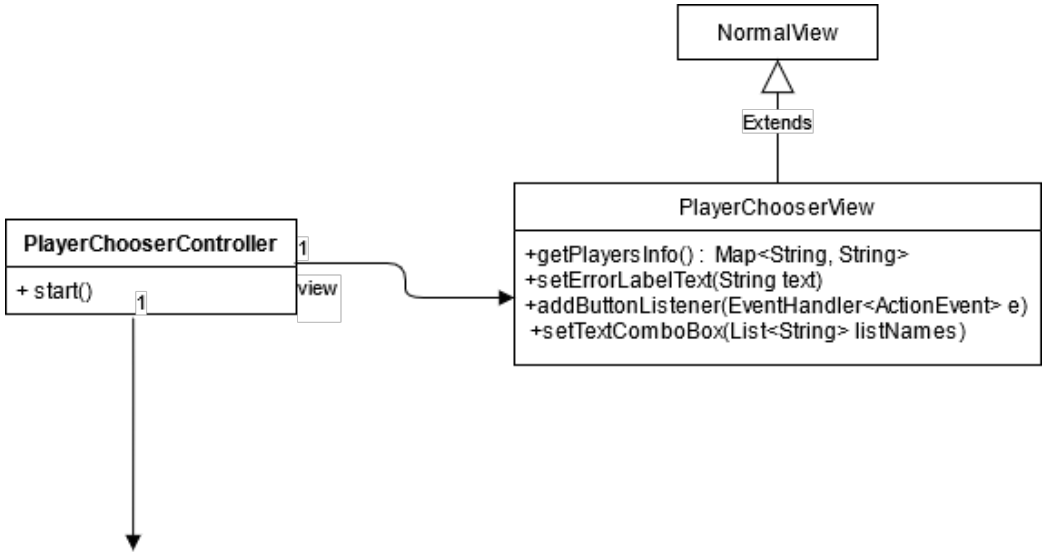


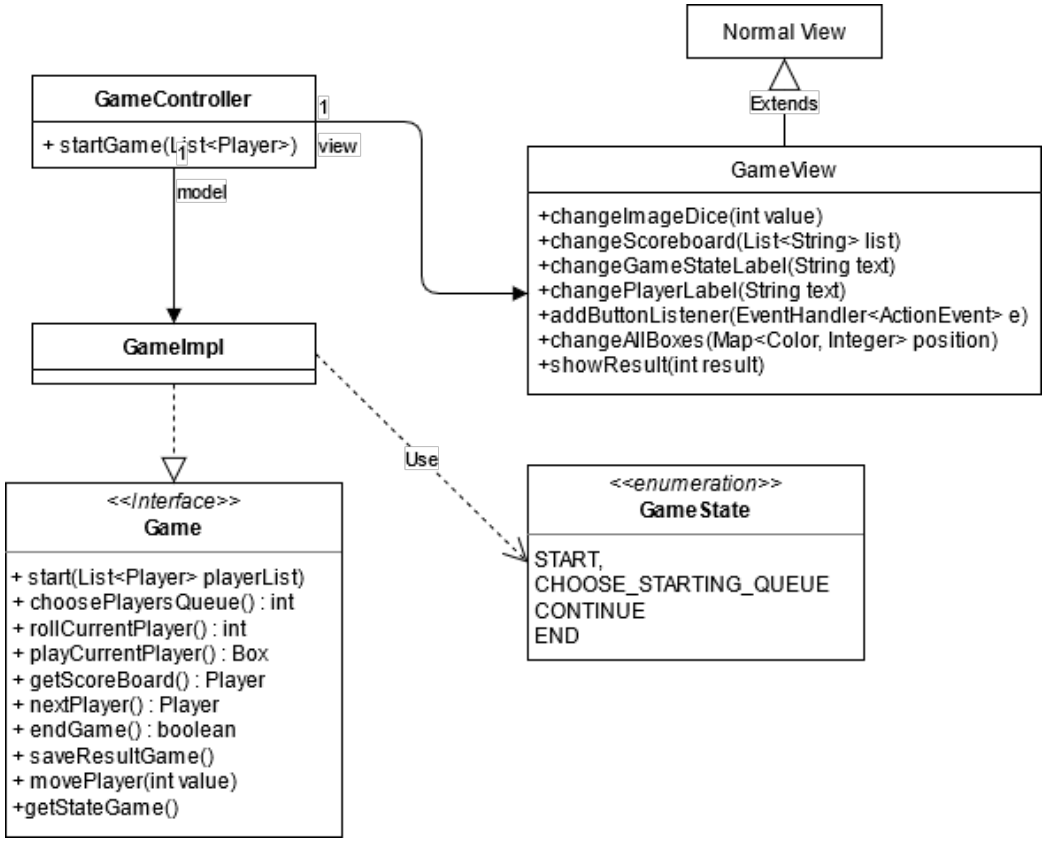
Figura 2.2: Rappresentazione UML delle relazioni tra entità che si occupano della creazione del campo

Gestione Gioco

- PlayerChooser: In questa fase si può scegliere il proprio nome e il proprio colore all'interno del gioco. E' possibile inserire i nomi dei giocatori sia tramite la ComboBox che legge tutti i nomi dei giocatori da un file che tramite il campo di testo sottostante. Alla pressione del pulsante verranno controllati che i nomi non siamo nulli, che non vi siano duplicati e che siano almeno due per poter creare una partita. Tutti questi controlli verranno effettuati dal controller che si interfaccia con PlayerChooserView.
  Rispettati questi vincoli verrà passata la lista dei giocatori al gioco principale.



- Game: Ricevuta la lista dei giocatori verranno inizializzate tutte le proprietà del gioco, a partire dall'enum che viene impostata come "START" fino alla creazione del campo di gioco. Fino a che tutti i giocatori non avranno lanciato almeno una volta il dado si sarà nella modalità "choose starting queue" dove verranno salvati i lanci di ogni giocatore per poi creare la sequenza di gioco a seconda di essi. Fatto ciò si passerà alla fase di gioco dove a turno ogni giocatore lancerà il dado e nel caso si andasse su di una casella di tipo minigioco verrà fatto partire il minigioco rilevato. Da qui si aspetterà che il giocatore termini il minigioco per poi farlo avanzare o retrocedere. Quando un giocatore si troverà esattamente sull'ultima casella si cambierà vista e si salverà la classifica dei giocatori su file.



FileUtility

Questa classe è stata implementata sfruttando la libreria GSON per salvare e caricare informazioni dai file json Per poter salvare qualsiasi tipo di oggetto ho pensato di realizzare ad una classe generica. Essa Espone solamente due metodi che si occupano di tutto il lavoro.

Se esiste il file la scrittura può essere effettuata sia in append che sovrascrivendo le informazioni e impedendo le duplicazioni di oggetti mentre la lettura è stata realizzata soprattutto grazie al typetoken offerto dalla libreria che rileva il tipo di oggetto che verrà letto da file.

View

Per ridurre la quantità di codice uguale per la realizzazione delle viste ho implementato il concetto di ereditarietà. La classe padre crea lo stage, caricando le informazioni a seconda del valore dell'enumerazione che raccoglie all'interno di sè le informazioni distintive di ogni vista: il loro nome e il file fxml che deve essere caricato.

I due figli differiscono nell'impossibilità di chiudere la finestra e nell'attesa di

un valore di ritorno. Queste classi sono state create come base da utilizzare per tutte le viste all'interno del gioco. Le schermate normali come la schermata iniziale estenderanno la vista basica mentre la seconda verrà utilizzata per tutte le schermate dei minigiochi in modo tale che non sia accessibile il campo di gioco mentre si svolge il minigioco e che ques'ultimo non si possa chiudere se non viene terminato prima.
Questa realizzazione permetterà l'aggiunta di altre schermate molto più semplicemente.

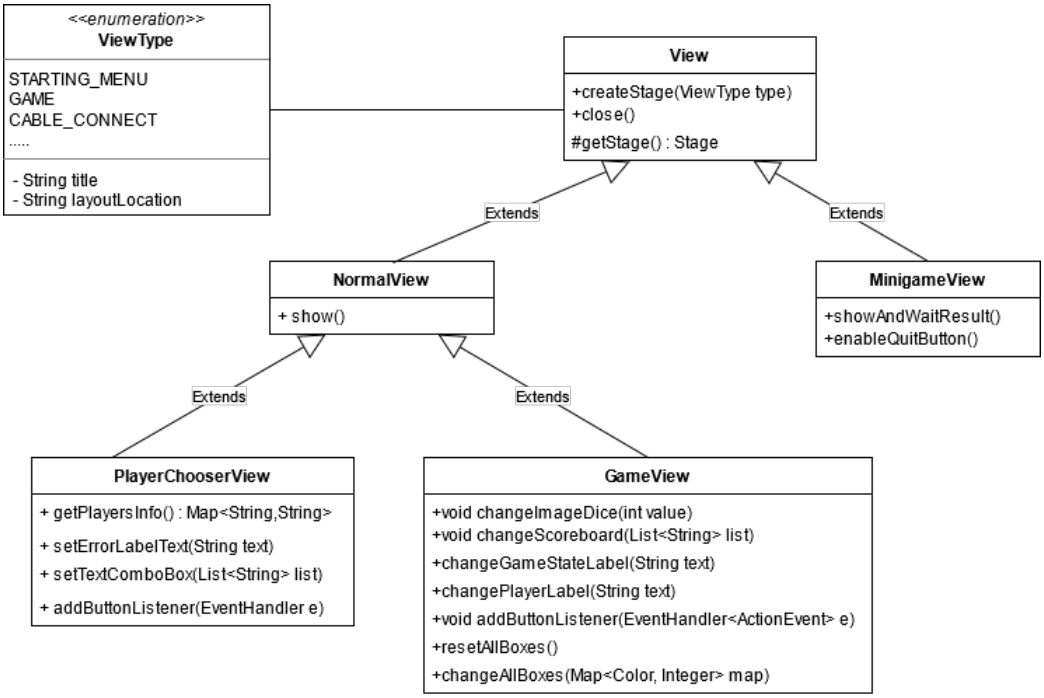
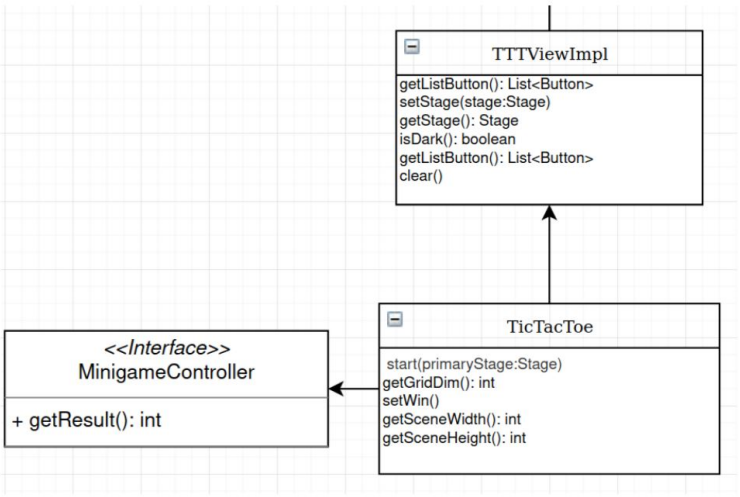


Figura 2.3: Rappresentazione UML della view implementata tramite ereditarietà

Davide Denicolò

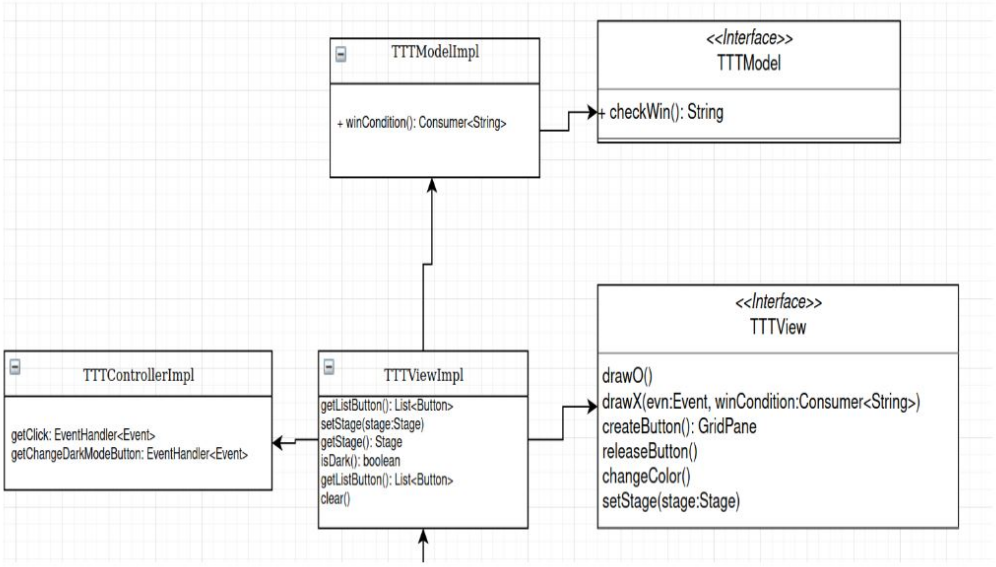
Il mio contributo è stato dato nella creazione di tre mini-giochi che si vanno a unire ad altri mini-giochi creati dai colleghi.
Il primo mini-gioco che illustro è Tic-Tac-Toe. Esso è stato progettato con JavaFX facendo fede al modello MVC.
C'è una classe principale chiamata **TicTacToe**: Essa gestisce la creazione della View e della schermata principale di gioco, verrà inizializzata una scena



di dimensioni Width e Height. Poi è presente un metodo usato per tornare il risultato della partita che verrà successivamente usato per lo score totale della partita a GooseGame.

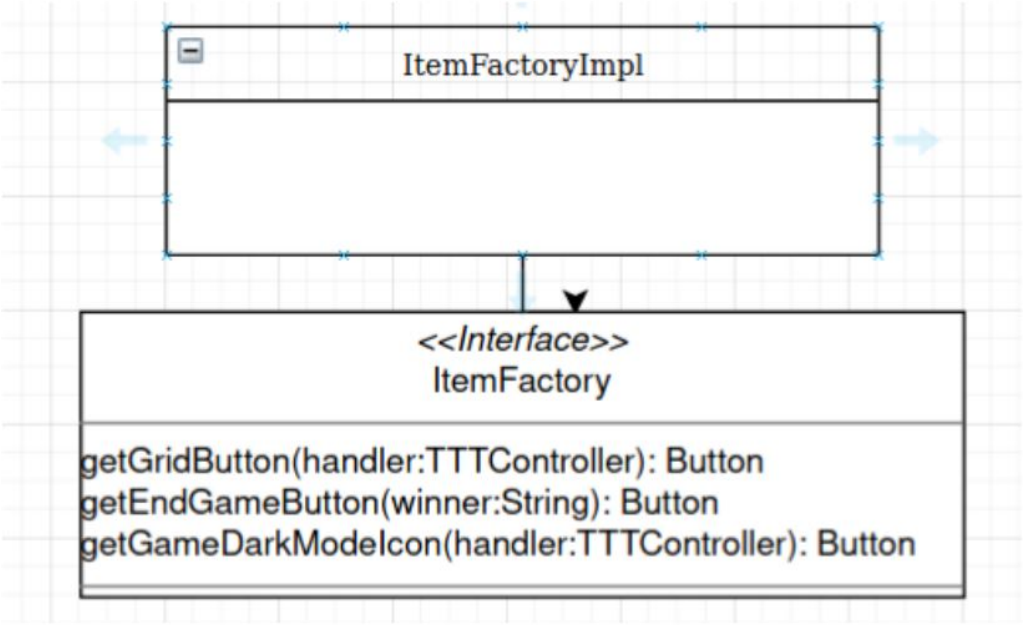
View:
La view implementa un interfaccia chiamata TTTview che modella le funzioni della TTTViewImpl. La view come descritto nel paradigma MVC gestisce la grafica del gioco. Come prima cosa popola le liste contenenti i bottoni nel gioco e inserendoli in una griglia a scelta tra (3x3, 4x4, 5x5, 6x6) e restituendola alla classe che implementa la TTTView, ovvero la classe madre TicTacToe. Dopo di che, una volta creata la griglia di gioco, essa cambia il testo dei bottoni in base al turno. A ogni click viene chiamata la Model che controlla se c'è un vincitore o meno, e lo comunica alla View, che continuerà a disegnare X e O sui bottoni.

Dopo di che, una volta creata la griglia di gioco, essa cambia il testo dei bottoni in base al turno. A ogni click viene chiamata la Model che controlla se c'è un vincitore o meno, e lo comunica alla View, che continuerà a disegnare X e O sui bottoni.
Model:
La model si occupa esclusivamente della logica del gioco, essa contiene un algoritmo nella funzione checkWin che calcola l'eventuale vincitore della partita.
Se c'è un vincitore: chiude lo stage principale, e avvia l'EndGameThread:
Sostanzialmente l'EndgameThread si avvia, e mostra il vincitore, dopo 1 secondo, il thread si chiude automaticamente. Per effettuare questo effetto è

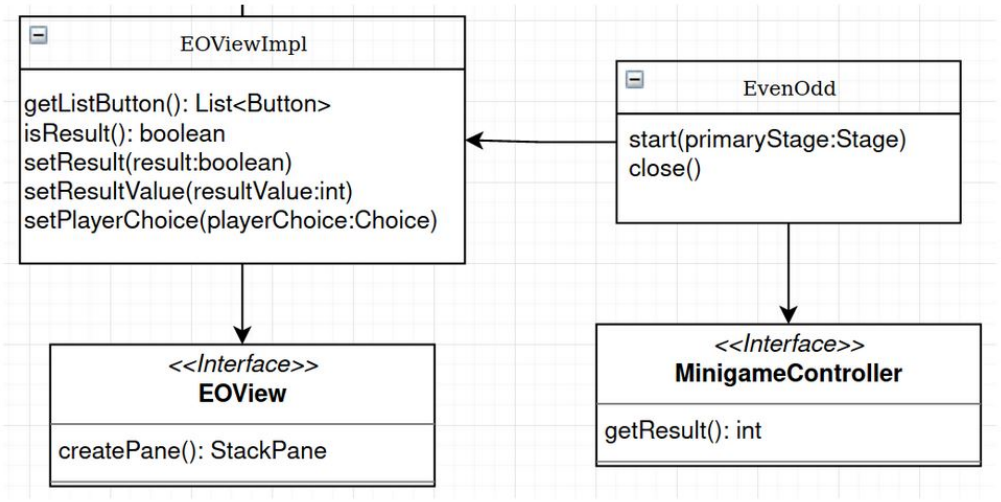

TTTModelImpl
+ winCondition(): Consumer<String>
<<Interface>>
TTTModel
+ checkWin(): String
TTTControllerImpl
getClick: EventHandler<Event>
getChangeDarkModeButton: EventHandler<Event>
TTTViewImpl
getListButton(): List<Button>
setStage(stage:Stage)
getStage(): Stage
isDark(): boolean
getListButton(): List<Button>
clear()
<<Interface>>
TTTView
drawO()
drawX(evn:Event, winCondition:Consumer<String>)
createButton(): GridPane
releaseButton()
changeColor()
setStage(stage:Stage)

stata usata una FadeTransition e una Task, che è avviata in un thread a parte, essa aspetta 1 secondo e dopo chiude lo stage.
Controller:
Il controller gestisce l'input utente, comunicando con la view per notificarla della scelta dell'utente di un'azione compiuta. Click, esso chiama la funzione drawX(). La View poi disegnerà la X e controllerà il vincitore. ChangeDarkModeButton: Chiama la funzione della View changeColor() e ne cambia il colore.

Come ultima funzione per ottimizzare la creazione degli item e delle immagini utilizzate, sono state create due classi, una contenente tutte le immagini in delle variabili statiche. Pertanto ogni qualvolta si utilizzerà un'immagine si farà riferimento a quest'ultima. Stesso discorso per i bottoni, la ItemFactory fornisce personalizzati utili all'applicazione.



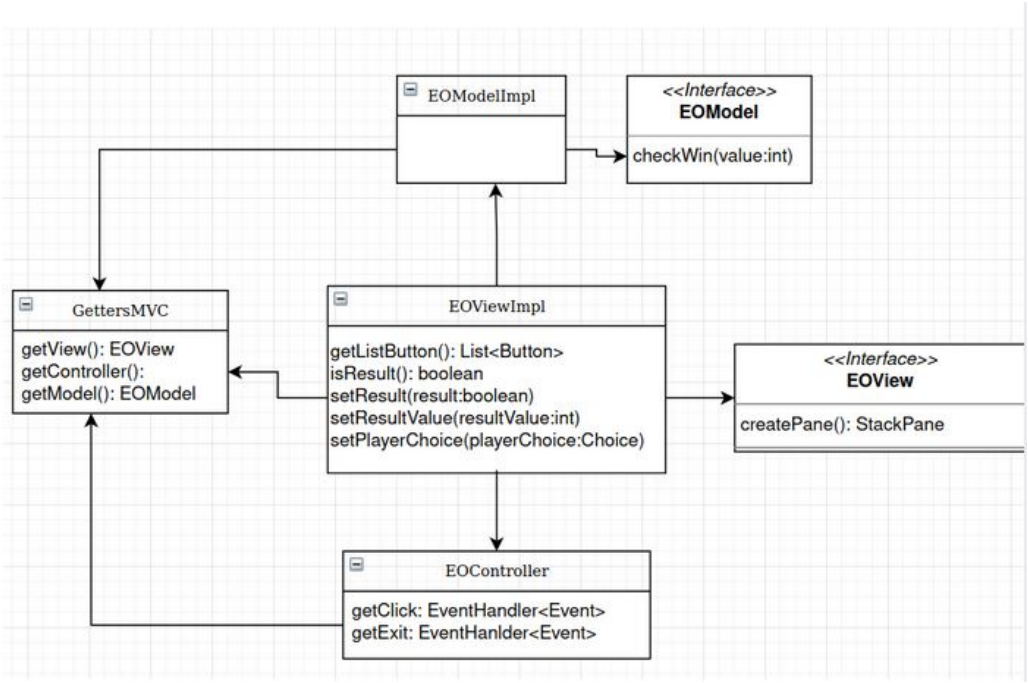
EVEN ODD
EvenOdd è stato creato con JavaFX utilizzando il paradigma MVC.
Come precedentemente svolto nel TicTacToe la classe EvenOdd ha anch'essa il compito di inizializzare la view; implementa poi un'interfaccia chiamata MinigameController che ha al suo interno una funzione getResult() che restituisce il punteggio ottenuto al mini-gioco. La EOViewImpl inizializza i componenti grafici implementando un'interfaccia chiamata EOView.



L'implementazione della view:
Il compito della view è quello di: creare i 3 bottoni nel gioco: "Pari", "Dispari", "Esci", (quest'ultimo si attiverà una volta finito il gioco), ed infine, cambiare l'immagine del vincitore in base al risultato ottenuto dalla Model. Pertanto c'è una comunicazione tra la view e il Model, le quali lavorano così: View → "scelta utente" al Model, Model → "vittoria o sconfitta" alla View.
Model:
Essa grazie alla classe GettersMVC riesce a comunicare con la View il proprio risultato, Vincente o Perdente. Opera in maniera molto semplice, genera un numero casuale e se coincide con la scelta 1(Dispari) 2(Pari) dell'utente allora si vince, si perde altrimenti.

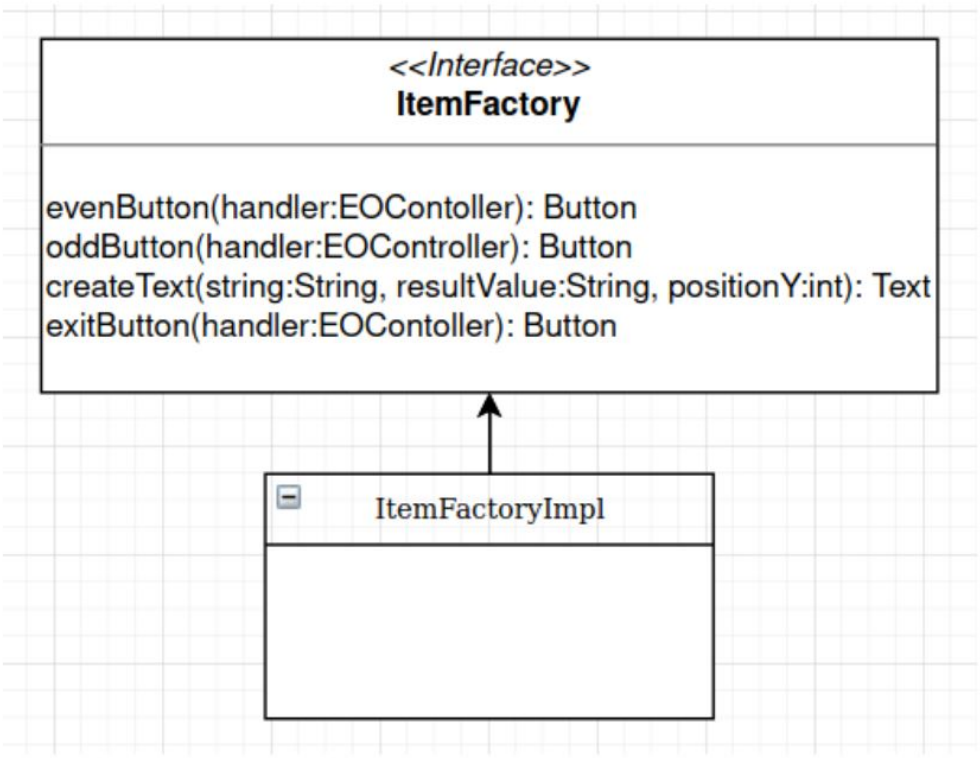
Controller:
I pulsanti ai quali affidare gli handler sono due, 1 i bottoni di scelta "pari" o "dispari", l'altro è il bottone "esci". Click, alla pressione del mouse su uno dei due bottoni a scelta, viene comunicato al Model la scelta utente 1(Dispari) 2(Pari), dopo di che vengono disabilitati i due bottoni e attivato il bottone "Esci". Exit, Se si clicca sul bottone "esci" verrà chiamata la funzione close().

Infine per comodità di realizzazione, sono state create due classi interamente responsabili delle: immagini e item. La prima classe , BackgroundLoader si occupa di fornire costanti statiche contenenti immagini di opportuna grandezza; la seconda, ItemFactoryImpl fornisce tutti gli item indispensabili per l'applicazione, come i bottoni.



SpaceShooter:
Il gioco space shooter è stato realizzato con JavaFX.
Il cuore portante di questo gioco è la classe principale, SpaceShooter. Essa estende l'interfaccia MinigameController la quale obbliga ad implementare la funzione getResult() utilizzata per restituire il risultato ottenuto a fine partita.
La classe SpaceShooter è cosi strutturata:

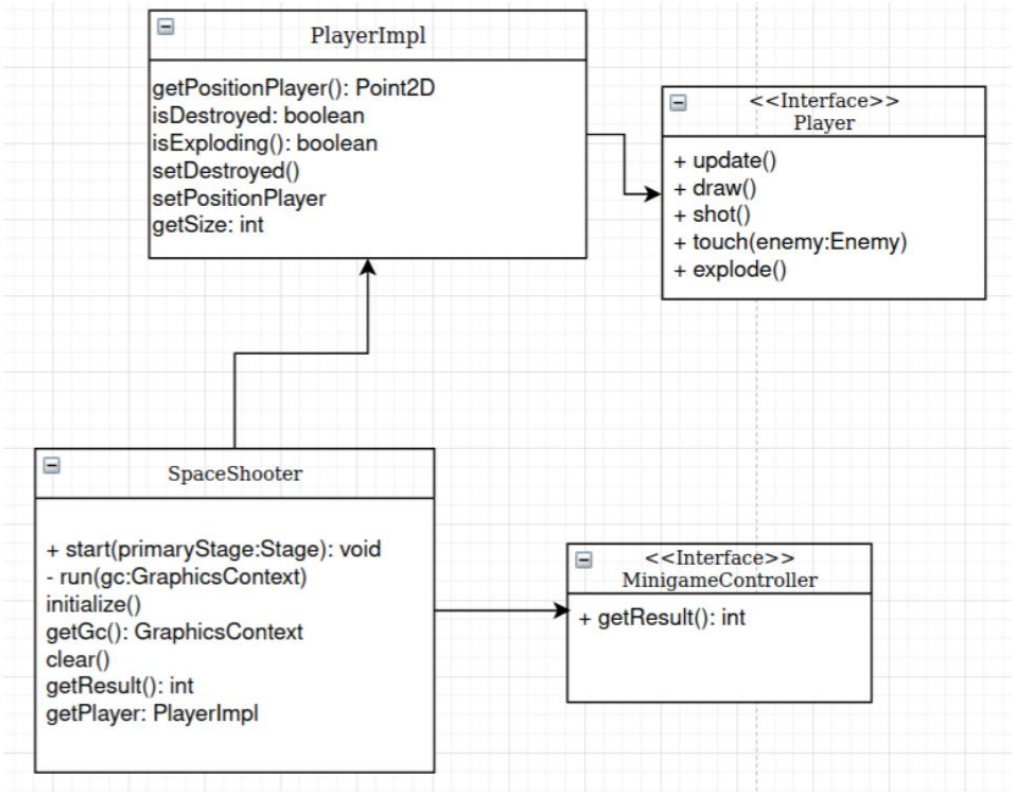
- Sono presenti 3 entità principali. Nemici, giocatore e proiettili, essi NON sono trattati come thread distinti, ognuno di queste entità è guidata da un thread principale.
- Canvas che crea la schermata di gioco delle dimensioni (Width e Height) presenti in InfoGame. Esso servirà anche per gestire i click del mouse qualora bisogna sparare un proiettile, e quando si dovrà cliccare sul tasto di exit.
- GraphicsContext che serve per reinderizzare le immagini tre entità sopra descritte più dei Text che fanno parte del gioco.
- TimeLine per eseguire in maniera contigua la funzione run() che aggiorna tutti i componenti dell'applicazione; l'aggiornamento viene effettuato facendo un forte uso di stream e for:each



- InfoGame che è che una classe che fornisce costanti pubbliche statiche per un loro utilizzo semplice da parte di tutte le classi di gioco/testing.
- Utilizzo della funzione run per aggiornare tutte le entità utilizzando i loro metodi, draw, update, touch ecc.

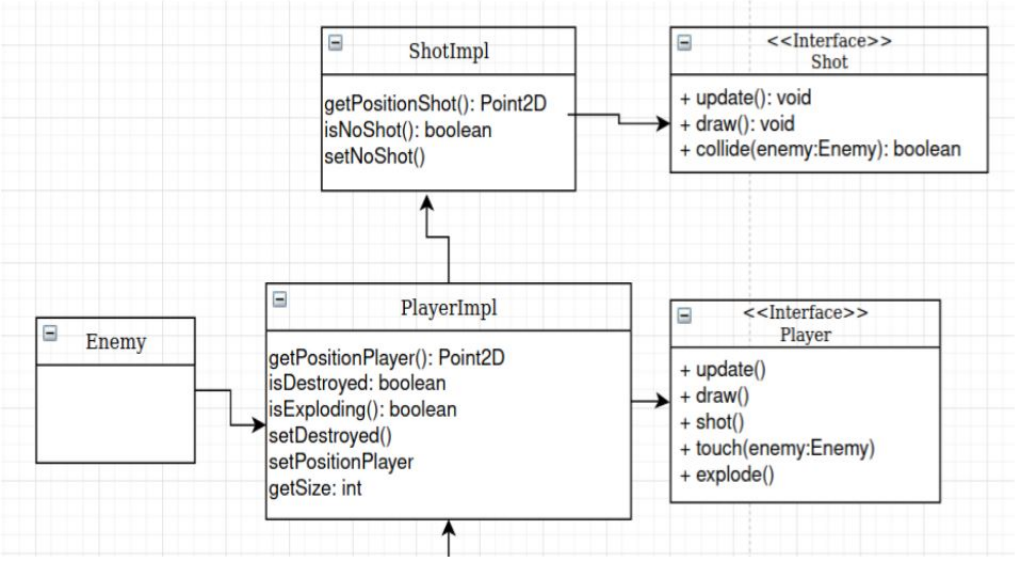
Le 3 entità del gioco per memorizzare le proprie coordinate viene utilizzata una libreria di JavaFX, chiamata geometry, che permette di implementare un oggetto chiamato Point2D che mi facilita la memorizzazione di punti dello spazio (le coordinate X,Y). In tal modo per muovere il player mi basterà prendere la posizione X del mouse e modificare il Point2D.
Per quanto riguarda l'immagine che rappresenta il player faccio riferimento alla classe InfoGame che mi fornisce l'immagine voluta.

Player
Questa è la classe madre per le entità, infatti la classe Enemy estende da Player.
Questa classe, poi, implementa l'interfaccia Player che mi fa usare update, draw, shot, touch e explode. Questi metodi sono il cuore del'entità, infatti



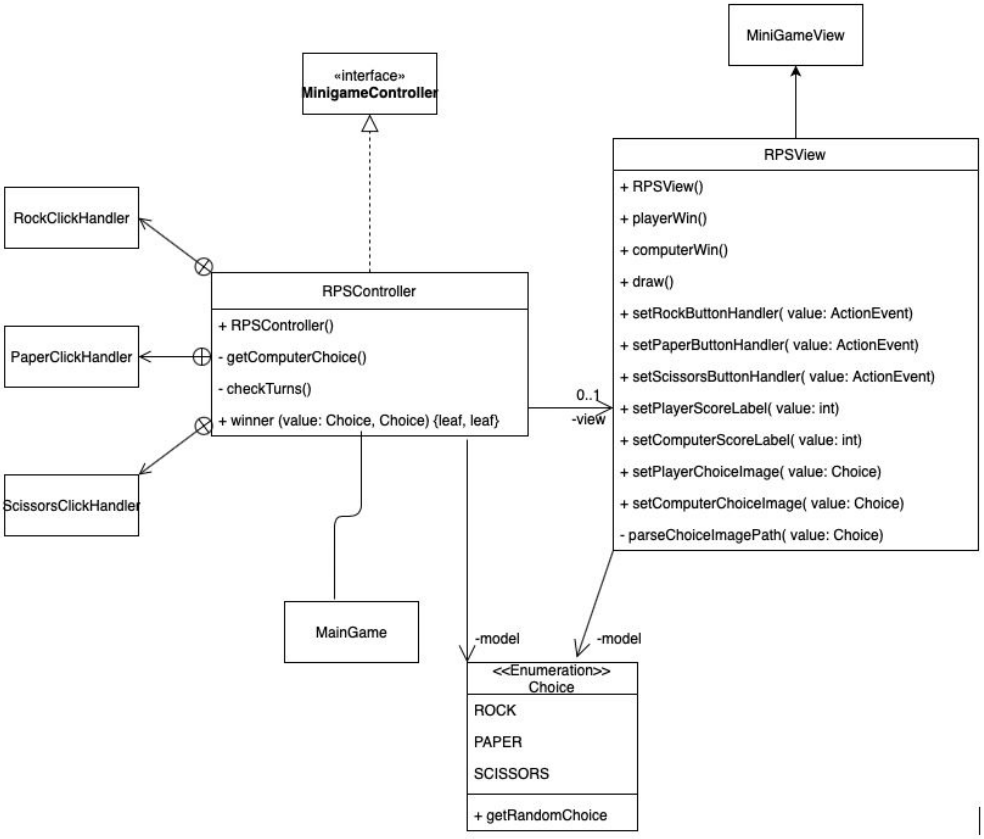
la permettono il movimento e la collisione della stessa.
La classe Enemy estende poi da Player:
Con la differenza che la velocità (coordinata Y) cambia in base allo score e si autodistrugge se non colpisce niente arrivando alla fine dello schermo.

Shot:
Il proiettille aggiorna la propria posizione verso l'alto, in senso opposto al nemico, infatti il suo compito è quello di colpirlo, a tal proposito la funzione collide verifica proprio ciò. La draw utilizza il graphics context per disegnare l'ovale nella sua posizione attuale.



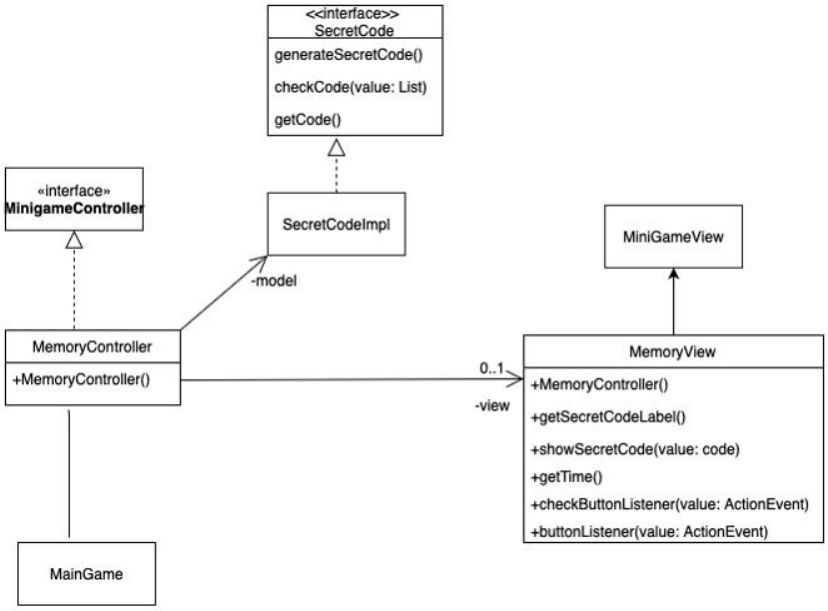
Endri Domi

RockPaperScissors: Il classo gioco rivisitato con aspetti grafici creati con JAVAFX e SceneBuilder.
Questo mini-gioco restituisce: un bonus calcolato moltiplicando il numero di vittorie al quadrato. Il mini-gioco viene svolto in tre turni, dove ogni turno finisce con: vittoria player, vittoria computer o pareggio. Si è deciso che in caso di pareggio il turno non verrà aggiornato (al meglio dei tre). La scelta del computer è gestita casualmente tramite il metodo getRandomChoice(). La scelta verrà reimpostata in ogni turno. Vi sono presenti le label per il punteggio dell'utente e del computer; vi è inoltre, una ulteriore label che restituisce un messaggio di vittoria, pareggio o sconfitta. Le scelte sia del player che del computer sono gestite tramite enumerazioni "Choice": ROCK, PAPER, SCISSORS. Una volta terminata la partita si abiliterà il pulsante "quit".

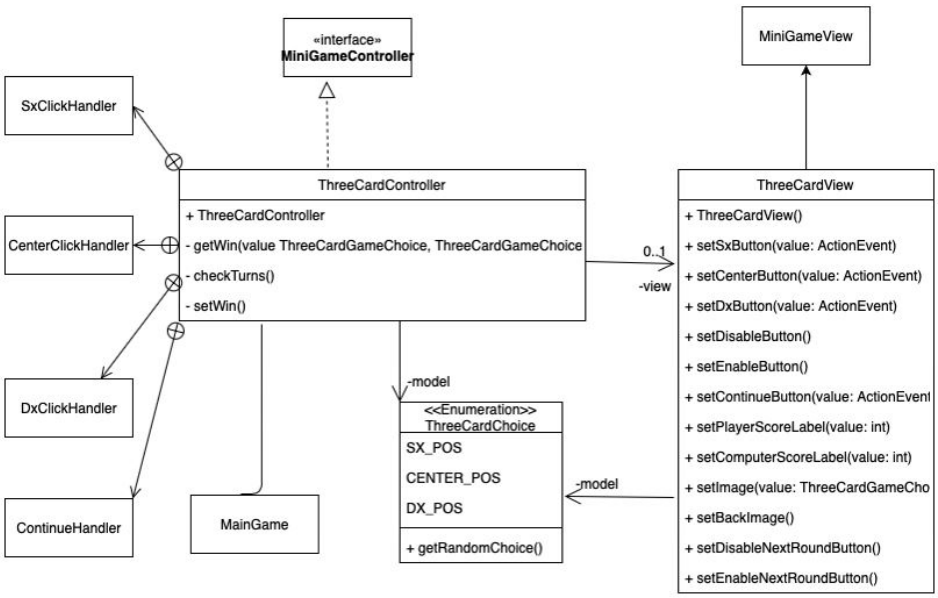


Memory:
Lo scopo del minigioco è quello di ricordare un "codice" che "scomparirà" dopo qualche secondo, quindi verrà impostato come "*****". L'utente attraverso una griglia contente i numeri dall'uno al nove, dovrà cercare di premere i pulsanti corrispettivi al "codice". Il "codice" viene generato casualmente dalla classe SecretCode, inoltre ne controlla gli errori e converte il codice sotto forma di stringa. Il codice è composto da una lista di cinque interi come il Pin del Bancomat. Alla fine dell'immissione da parte del player, quest'ultimo dovrà premere il pulsante "Check" il quale controllerà l'effettiva validità del "codice" e confronterà tramite il metodo checkCode(List¡Integer¿ inputCode) gli errori. Per gestire l'immissione da parte dell'utente (quindi della pressione dei nove pulsanti dedicati a questo ruolo), viene svolta attraverso un confronto tra due liste: "secretCode" e "inputCode", dunque ne verifica

la corrispondenza. Questo mini-gioco restituisce: un bonus/malus calcolato in base ai secondi rimanenti meno gli errori commessi dall'utente.
Al termine della partite si abiliterà il pulsante "quit".



Three-Card-Game:
Questo mini-gioco si ispira al gioco delle tre carte dove solo una su tre è quella giusta. La scelta della carta giusta è fatta casualmente dal metodo dell'enumerazione: getRandomChioce(). Il player dovrà tentare di indovinare le carte giusta premendo il pulsante che si trova sotto di essa. Anche questo mini-gioco si svolge ai meglio di tre. Al termine del turno il player dovrà premere il pulsante "continue", mentre tutti gli altri pulsanti si disabiliteranno. In ogni turno il mini-gioco reimposta la carta coperta e cambia scelta, sempre casualmente.


«interface»
MiniGameController
MiniGameView
SxClickHandler
ThreeCardController
+ ThreeCardController
- getWin(value ThreeCardGameChoice, ThreeCardGameChoice
- checkTurns()
- setWin()
ThreeCardView
+ ThreeCardView()
+ setSxButton(value: ActionEvent)
+ setCenterButton(value: ActionEvent)
+ setDxButton(value: ActionEvent)
+ setDisableButton()
+ setEnableButton()
+ setContinueButton(value: ActionEvent
+ setPlayerScoreLabel(value: int)
+ setComputerScoreLabel(value: int)
+ setImage(value: ThreeCardGameCho
+ setBackImage()
+ setDisableNextRoundButton()
+ setEnableNextRoundButton()
CenterClickHandler
0..1
-view
DxClickHandler
-model
<<Enumeration>>
ThreeCardChoice
SX_POS
CENTER_POS
DX_POS
+ getRandomChoice()
-model
ContinueHandler
MainGame

Capitolo 3

Sviluppo

3.1 Testing automatizzato

La realizzazione del testing automatizzato è stato portato a termine grazie all'utilizzo di due framework diversi, uno per le classi non grafiche, JUnit 5 e il secondo per l'interfaccia grafica implementata tramite JavaFX, TestFX.

Leonardo Carboni

Cable connect:
Sviluppo di un test automatizzato tramite l'ausilio della libreria TestFX che crea un'istanza della view del minigioco e tramite un robot svolge i vari test di collegamento dei cavi, verificando se è possibile cliccare o meno i vari bottoni rappresentanti "il socket di inizio e fine del cavo"

PhraseCatch:
Verifica del corretto numero di errori in due frasi diverse.

Andrea Ongaro

File Utility Test:

- Confronto della lista che viene salvata su file con quella letta da file. Testato con classi diverse: PlayerImpl e String.
- Controllo delle due metodologie di salvataggio sul file: sovrascrizione e aggiunta in coda.
- Controllo del lancio di eccezioni nel caso in cui si provi a leggere un file che non esista.

Game Test:

- Controllo che le posizioni vengano inizializzate all'inizio del gioco.
- Controllo che il gioco finisca quando un giocatore raggiunge il limite del campo di gioco. In questa funzione di testing è stato fatto avanzare solo un giocatore controllando solo la sua posizione finale.

Queue Test:

- Controllo che la coda dei giocatori continui all'infinito rispettando la sequenza della lista, iterando per un numero molto alto di volte la sequenza e controllando che coincida con il giocatore atteso.
- Controllo creazione della sequenza di partenza dei giocatori calcolata a seconda dei valori dei dadi lanciati dai giocatori.

Board Test:

- Controllo creazione del campo di gioco.
- Controllo che venga lanciata l'eccezione in caso di una dimensione troppo piccola per poter creare il campo di gioco.

Davide Denicolò

Per il testing dei tre mini-game Tic tac toe, even odd e space shooter è stato usato esclusivamente JUnit 5 optando per un controllo logico e non grafico.

- Tic tac toe: Verifica il vincitore , che sia il giocatore, il computer o un pareggio.
- Even Odd: controllo che restituisca un risultato corretto nei casi previsti dal gioco.
- Space shooter: verifica delle collisioni facendo variare le X e Y delle entità.

Endri Domi

RockPaperScissorsTest:

- Verifica il vincitore.

Memory:

- Verifica del corretto conteggio degli errori.

ThreeCardGame:

- Verifica sia la vittoria che la sconfitta.

3.2 Metodologia di lavoro

La divisione iniziale dei lavori e delle parti non è stata rispettata appieno. Durante lo sviluppo, infatti, ci siamo resi conto che sarebbe stato più opportuno modificare leggermente la separazione dei lavori per ottimizzare e velocizzare la realizzazione rispettando sempre il fatto che ognuno avesse della logica applicativa all'interno della propria parte.
Per la realizzazione di questo progetto è stato utilizzato il DVCS Git, il quale essendo distribuito permette ad ogni collaboratore di mantenere un proprio repository locale che contiene tutti i file e metadati presenti nel repository principale. I branch sono stati utilizzati nel seguente modo:

- Main: rilascio della versione stabile del progetto.
- Develop: sviluppo della logica di gioco.

Inoltre, per separare maggiormente il lavoro di ognuno, si è pensato di spostare la costruzione del proprio minigioco su un branch dedicato così da essere ancora più indipendenti. A livello client alcuni studenti hanno utilizzato Git Desktop mentre altri Git Bash.

Leonardo Carboni

Realizzazione file FXML (JavaFX) di Menu, Player Chooser, How to Play, Main Game, Minigiochi Cable Connect e Phrase Catch e WinScreen (e i rispettivi css). Realizzazione loghi e mockup di experience design.

Implementazione logica dei due minigiochi sopracitati, menu, Win Screen, piccola parte di main game. Gestione completa della classe countdown (per i minigiochi) e duration (per il gioco principale) I file realizzati si trovano in:

- Model.phrase, model.Duration, model.colors
- Controller.menu, Controller.howtoplay, Controller.phrasecatch, controller.playerchooser, controller.cableconnect, controller.winscreen
- Utility.countdown
- View.menu, view.cableconnect, view.phrasecatch, view.winscreen view.howToPlay
- Resources/layouts

Andrea Ongaro

Realizzazione della logica di Gioco, Gestione IO e della visualizzazione dei menu e delle viste. I file realizzati si trovano all'interno dei package

- model.board, model.box, model.dice, model.game, model.player, model.queue, model.rank.
- controller.game,controller.playerchooser
- view.game, view.playerchooser e i file view, normalView, minigameView e ViewType
- utility.file

Davide Denicolò

Realizzazione dei minigiochi: tic-tac-toe, even odd, I suddetti mini-giochi si trovano in:

- application.minigame.evenodd.*
- application.minigame.tictactoe.*
- application.minigame.spaceshooter.*
- resources.evenodd, tictactoe e spaceShooter

Endri Domi

Realizzazione file FXML di: Rock-Paper-Scissors, Memory e Three-Card-Game. Implementazione dei minigiochi sopra citati. I file realizzati si trovano in:

- Model.SecretCode, Model.Choice.ThreeCardChoice, Model.Choice.RockPaperScissorsChoice.
- View.MemoryView, view.RPSView, view.ThreeCardView
- Controller.MemoryController, Controller.RPSController, Controller.ThreeCardController
- Resources/layouts

3.3 Note di sviluppo

Leonardo Carboni

- Ampio utilizzo di javafx e scenebuilder
- Utilizzo di testFX
- Implementazione minigiochi
- Gestione countdown e duration
- Utilizzo di elementi di grafica dinamica per javafx
- Creazione progetto e utilizzo di gradle

Andrea Ongaro

- Utilizzo del Libreria GSON. Gson è una libreria Java open source per serializzare e deserializzare oggetti Java su JSON.
- Creazione di una classe generica, implementata, dopo essermi bloccato, grazie ad una risposta su sito stackoverflow
  `https://bit.ly/3sVUGPS`
- Minimo utilizzo della libreria JAVAFX

Davide Denicolò

- Utilizzo della libreria JavaFX: usata per modellare aspetti essenziali come il Canvas per intercettare i click dell'utente, il GraphicsContext utile per creare animazioni, Timeline per gestire i frame e Bottoni utili al fine di creazione dei minigame.
- Utilizzo di JUnit: usato per testare la logica del gioco, collisioni, algoritmo usato per vedere il vincitore nel tris e nel mini-game pari o dispari
- Utilizzo di Stream: usata per aggiornare efficacemente entità come i nemici e i proiettili in SpaceShooter
- Utilizzo di Gradle: per rimanere aggiornati come team delle nuove librerie

Endri Domi

- Utilizzo di javafx e scenebuilder
- Implementazione mini-giochi
- Test dei mini-giochi

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

- Possibilità di una più ampia caratterizzazione dei personaggi, magari con diversi dadi a seconda del tipo di utente.
- Maggior numero di minigiochi
- Realizzazione di una grafica 3D, ispirandosi alla grafica di Monopoly Plus"[1]

4.1.1 Leonardo Carboni

Mi ritengo la parte trainante del gruppo. Ho creato il progetto dopo aver studiato gradle, terminato lo sviluppo della mia parte prima di tutti e aiutato i miei compagni nella realizzazione delle loro parti. Ho creato la maggior parte dei file fxml e deciso il design del gioco eseguendo prima un mockup. Le parti che sono risultate più difficili sono sicuramente riguardanti l'utilizzo del pattern mvc e di gradle, soprattutto la gestione dei file nella cartella resources al momento di creazione per via del funzionamento della java virtual machine. Sono comunque soddisfatto dei miei risultati poiché ho imparato come si sviluppa un progetto su "grossa" scala in gruppo.

4.1.2 Andrea Ongaro

Personalmente direi che le parti di forza del codice (realizzata da me) all'interno del progetto sono la classe che si occupa di leggere e scrivere su un file

[1] `https://www.ubisoft.com/it-it/game/monopoly/monopoly` videogioco strategico sviluppato da Asobo Studio e pubblicato da Ubisoft nel 2014

e la classe dedicata alla coda di giocatori. Sono riuscito ad utilizzare una libreria esterna (GSON) e ad implementare il tutto come una classe generica così da poter essere utilizzata per poter salvare diversi tipi di oggetti. Pur essendo Un lato negativo di questa classe purtroppo è la realizzazione dell'append e della ricerca di contenuti già importanti all'interno del file per il quale non sono riuscito a trovare un metodo più efficiente. Oltre a questa classe PlayerQueue per la creazione di un iteratore infinito, punto base per questo gioco dove non i giocatori si devono susseguire uno dopo l'altro senza nessuna eccezione. Infine, per quanto riguarda l'MVC, sono parzialmente soddisfatto della sua integrazione nel gioco, sicuramente si sarebbe potuto fare meglio, ma come prima esperienza sono contento.

4.1.3 Davide Denicolò

Il lavoro è proceduto a un buon ritmo, non ci sono stati problemi di assegnazione dei compiti e di coordinazione. Quindi per quanto mi riguarda ho svolto il mio lavoro in maniera fluida e solidale con quanto concordato. Come punto di forza del mio lavoro svolto ci sia la chiarezza e la semplicità con la quale è stato implementato, specialmente dei mini-giochi Tris e Pari o Dispari. D'altro canto la parte negativa che mi sento di evidenziare a onor del vero è che i mini-giochi sono semplici, non sono intrinsecamente complicati, questo, come è facile intuire, ha facilitato enormemente il lavoro da me svolto. Motivo per cui ho cercato di concentrarmi sulla qualità piuttosto che quantità.

4.1.4 Endri Domi

Come prima esperienza di lavoro in gruppo, che nonostante qualche incomprensione ed il fatto che l'assegnamento dei ruoli non sia stato rispettato a pieno, lo sviluppo tutto sommato è andato bene. Per quanto riguarda il mio lavoro, penso che i miei mini-giochi sono di natura semplice. Avrei preferito fare qualcosa di più costruttivo è stimolante. Tuttavia, per lo sviluppo della mia porzione di progetto ho cercato il più possibile di utilizzare il pattern MVC, come visto a lezione, al meglio delle mie possibilità. Ho creato i file fxml dei mini-giochi attraverso SceneBuilder. Per seguire una linea più coesa col gruppo, ho utilizzato classi innestate, per catturare gli eventi ed enumerazioni. Sono comunque contento di questa esperienza, in un futuro lavorativo le conoscenze acquisite sia a lezione che dai miei compagni di progetto (come l'utilizzo corretto del pattern e di gradle), potrebbero risultarmi utili.

4.2 Difficoltà incontrate e commenti per i docenti

4.2.1 Leonardo Carboni

Amplierei le spiegazioni riguardanti gradle e javafx (soprattutto l'integrazione con i pattern e la comunicazione tra vari stage / l'aggiornamento dei dati) mentre penso che durante le lezioni git venga trattato abbastanza approfonditamente così come i pattern a livello teorico. Il gruppo ha riscontrato a mio parere grossi problemi nel lavorare insieme, seguire la linea comune delineata e nell'uniformità generale del progetto

4.2.2 Andrea Ongaro

Ho riscontrato e ho notato all'interno del gruppo Whatsapp dedicato a questo corso, dei problemi con Gradle. Secondo me sarebbe molto utile che ci si soffermi un po' di più sulla realizzazione di un progetto tramite Gradle. A livello di gruppo, purtroppo, non c'è stata una grande comunicazione durante la realizzazione del progetto se non con il collega Carboni. Se avessimo collaborato durante il progetto come negli ultimi giorni sarebbe uscito un progetto più omogeneo e di più alto livello.

4.2.3 Davide Denicolò

Le difficoltà riscontrate sono per lo più di coordinazione. Tuttavia sono soddisfatto di come si è svolto il progetto.

4.2.4 Endri Domi

Personalmente, darei molto più peso all'utilizzo dei pattern, poiché vengono trattati perlopiù nelle lezioni di teoria, utilizzo di lambda e stream e nella lezione 15-GUI parlerei anche di JavaFx, sia nelle lezioni di teoria che di laboratorio. Ho apprezzato le lezioni su git, molto utili per lo sviluppo del progetto. Approfondirei invece l'utilizzo dell' UML a lezione. Il corso è stato interessante, consiglierei di non presentare il codice durante le lezioni, ma di svilupparlo insieme agli studenti, in modo tale da capirne il ragionamento.

Appendice A

Guida utente

L'utilizzo del software è molto semplice anche grazie alla sua interfaccia molto pulita.

Schermata Iniziale

Composta da tre semplici bottoni:

- Play Game: si verrà rimandati alla schermata successiva per la scelta dei giocatori.
- How to Play: Spiegazione del funzionamento di ogni minigioco.
- Exit : Uscita dal gioco

Scelta Giocatori

In questa schermata si potranno scegliere i nomi dei giocatori e il proprio colore all'interno del gioco. Si dovranno scegliere minimo due giocatori e ogni giocatore dovrà avere un nome univoco(diverso dagli altri giocatori). I nomi dei giocatori potranno essere scelti nelle comboBox dove appariranno i nomi precedentemente salvati da altri utenti o potranno essere scritti nelle textField sottostanti. Cliccando il bottone "Play" nel caso in cui venissero soddisfatti i requisiti si verrà portati nella schermata di gioco in caso contrario verrà visualizzato il motivo per cui non si è potuti andare avanti.

Schermata di gioco

Una volta in questa schermata si dovrà lanciare i dadi per poter designare quale sarà la sequenza dei giocatori che giocherà, la sequenza verrà calcolata a seconda del valore del dado, dal lancio più alto al lancio più basso.
Dopo che verrà creato questa lista, il primo giocatore potrà lanciare il dado e a seconda della casella in cui si arriverà verrà lanciato un minigioco e, a seconda dell'esito del gioco, si potrà andare più avanti o più indietro o non succederà nulla.

Spiegazione di ogni minigioco

SassoCartaForbici:
Le regole sono semplici: Il sasso batte le forbici, le forbici battono la carta e la carta batte il sasso. La partita viene svolta ai meglio dei tre, in caso di pareggio si procederà con un nuovo turno. Per esprimere la propria scelta basta cliccare su uno dei tre pulsanti, al termine della partita bisognerà cliccare il tasto "Quit".

Memory:
Lo scopo del gioco è quello di ricordare e inserire, tramite gli appositi pulsanti, una sequenza di 5 numeri che dopo un breve tempo verranno oscurati. Il mini gioco termina premendo il pulsante "check", successivamente il pulsante "quit".

ThreeCardGame:
Lo scopo del gioco è quello di indovinare dove si trova la "regina" (rappresentata da una spunta verde). Sono presenti tre carte: una di esse sarà la "regina", mentre le altre saranno i "due"(rappresentati da una X rossa). Al termine di ogni turno si abiliterà un pulsante "continue" che permetterà di effettuare un nuovo turno. Il gioco si svolge al meglio dei tre.
Il pari o dispari è un gioco di natura molto semplice, scegli se puntare su PARI o DISPARI tramite gli appositi bottoni, dopo di che, se la tua scelta corrisponde con quella del computer hai vinto, altrimenti hai perso...

Tic-Tac-Toe:
Il Tic-Tac-Toe (Tris) si gioca in due in una griglia 3x3. Un giocatore mette la X, l'altro (raffigurato dal computer) la O, alternandosi l'un l'altro.
Vince chi mette il proprio segno allineato in: obliquo, verticale e orizzontale. Nella speranza di renderlo competitivo si può giocare in 4x4,5x5,6x6.

Space-Shooter:
Lo space-shooter (Space Invader) consiste nello sparare ai razzi evitando che quest'ultimi distruggano la navicella. Per sparare un colpo basta cliccare il tasto sinistro del mouse.

Cable Connect:
Minigioco a tempo nel quale bisogna collegare i vari cavi colorati ai loro corrispettivi quadrati. Basta cliccare su uno dei bottoni colorati a sinistra e successivamente al corrispettivo a destra. Quando tutti i cavi sono collegati il tempo si ferma.

Phrase Catch:
Minigioco a tempo nel quale bisogna scrivere la frase mostrata a video nel minor tempo possibile evitando gli errori. Al termine dell'inserimento della frase basterà cliccare il tasto invio sulla tastiera o il bottone "Submit".

Fine Gioco

Una volta che un giocatore arriverà sull'ultima casella, verrà aperta una schermata con la classifica finale del gioco. In questa parte si potrà scegliere di giocare nuovamente venendo riportati nella schermata per la scelta dei giocatori o di chiudere il gioco.

Appendice B

Esercitazioni di laboratorio